(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平8－194719

(43)公開日　平成８年(1996)７月30日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　17/30					
		9194－5L	Ｇ０６Ｆ　15/ 40	３７０　Ａ	
		9194－5L	15/ 419	３２０	

審査請求　未請求　請求項の数12　ＯＬ　（全 17 頁）

(21)出願番号　特願平7－230843

(22)出願日　平成７年(1995)９月８日

(31)優先権主張番号　特願平6－281054

(32)優先日　平６(1994)11月16日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号

(72)発明者　難波　功
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

(74)代理人　弁理士　大菅　義之　（外１名）

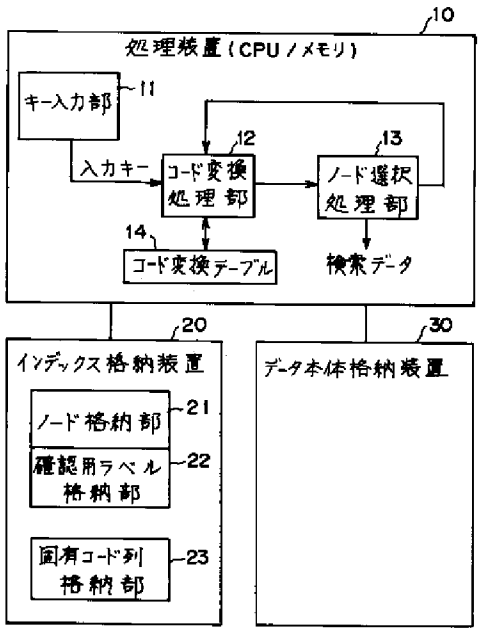
(54)【発明の名称】　検索装置および辞書／テキスト検索方法

(57)【要約】

【課題】自然言語の辞書やデータベース等のトライ法を用いる検索装置および辞書／テキスト検索方法に関し、検索の高速化とインデックスサイズの縮小化を実現することが課題である。

【解決手段】圧縮されたトライノードを格納するノード格納部２１と、検索対象となる入力キーの単位コードがノードに含まれるか否かを照合するための確認用ラベルを格納する確認用ラベル格納部２２と、トライ中のキー固有のコード列を格納する固有コード列格納部２３とを用意し、ノード格納部２１におけるノード要素を単位コードでたどりながら、確認用ラベル格納部２２および固有コード列格納部２３を参照して検索キーの有無を確認する。

本発明の構成例を示す図



【特許請求の範囲】
【請求項１】　キーを入力し、キーに対応するデータをトライ法により検索する検索装置において、
圧縮されたトライのノードを格納するノード格納部と、
圧縮されたトライのノードの照合の際に検索対象となる単位コードがノードに含まれるか否かを判定するためのトライのノードに対するラベルを格納するノード要素の確認用ラベル格納部と、
トライ中のキー固有のコード列を格納する固有コード列格納部と、
入力キーを１単位コードずつ取り出し、各単位コードをトライの内部コードに変換するコード変換処理部と、
単位コードがノードに含まれるか否かを前記ノード要素の確認用ラベル格納部を用いて確認し、単位コードがノードに含まれることを確認した場合に、前記ノード格納部から次のトライのノード情報または前記固有コード列格納部へのポインタ情報を得て、次の単位コードに対する照合処理に進むノード選択処理部とを備えたことを特徴とする検索装置。
【請求項２】　請求項１記載の検索装置において、
前記ノード格納部の各トライのノードには、次のトライのノードの起点となるポインタまたは前記固有コード列格納部内のキーの固有コード列へのポインタが格納され、
前記ノード選択処理部は、着目するトライのノードに次のトライのノードの起点となるポインタが格納されている場合には、その起点となるポインタと次の単位コードから得られた内部コードとによって定まるノードに前進し、着目するトライのノードに前記固有コード列格納部内のキーの固有コード列へのポインタが格納されている場合には、次の単位コード以降をポインタで示されるキーの固有コード列と照合するように構成されていることを特徴とする検索装置。
【請求項３】　請求項２記載の検索装置において、
前記ノード選択処理手段は、前記起点となるポインタの値に前記次の単位コードから得られた内部コードの値を加算して、次に着目すべきノードの位置を求め、該着目すべきノードに前記次の単位コードが対応するか否かを判定するように構成されることを特徴とする検索装置。
【請求項４】　請求項１記載の検索装置において、
入力キーとして使用される頻度の高い単位コードから順に並べたコード変換テーブルを格納するコード変換テーブル格納手段をさらに備え、
前記コード変換処理手段は、前記コード変換テーブルを用いて、使用頻度の高い単位コードほど順位の高い内部コードに変換するように構成され、前記ノード格納手段は、内部コードが表す順位に従って、対応するトライのノードを格納するように構成されることを特徴とする検索装置。
【請求項５】　請求項１記載の検索装置において、
前記確認用ラベル格納手段は、２進木を構成する複数のノード列のラベルを互いに重複しないように格納したラベル配列を格納するように構成され、前記ノード格納手段は、該ラベル配列に対応して前記次のトライの情報または前記固有コード列格納手段へのポインタ情報を格納するように構成されることを特徴とする検索装置。
【請求項６】　請求項１または請求項２記載の検索装置を用いた辞書検索方法であって、
自然言語の単語をキーとして圧縮されたトライを構成し、
自然言語の単語に対応するデータを、圧縮されたトライを用いて検索することを特徴とする辞書検索方法。
【請求項７】　請求項１または請求項２記載の検索装置を用いた辞書検索方法であって、
自然言語の音声データをキーとして圧縮されたトライを構成し、
自然言語の音声データに対応するデータを、圧縮されたトライを用いて検索することを特徴とする辞書検索方法。
【請求項８】　請求項１または請求項２記載の検索装置を用いたテキスト検索方法であって、
テキストデータをキーとして圧縮されたトライを構成し、
テキストデータに対応するデータベースのテキストを、圧縮されたトライを用いて検索することを特徴とするテキスト検索方法。
【請求項９】　キーを入力し、キーに対応するデータをトライ法により検索する検索装置における記憶媒体であって、
検索対象となる単位コードがノードに含まれるか否かを判定するためのラベルを含む圧縮されたトライのノードを作成する手段と、
トライ中のキー固有のコード列を表す固有コード列を作成する手段と、
入力キーを１単位コードずつ取り出し、各単位コードをトライの内部コードに変換する手段と、
単位コードがノードに含まれるか否かを前記ラベルを用いて確認する手段と、単位コードがノードに含まれることを確認した場合に、前記トライのノードから次のトライのノード情報と前記固有コード列へのポインタ情報のうちの１つを得て、次の単位コードに対する照合処理に進む手段とを備えることを特徴とする記憶媒体。
【請求項１０】　キーを入力し、キーに対応するデータをトライ法により検索する方法において、
検索対象となる単位コードがノードに含まれるか否かを判定するためのラベルを含む圧縮されたトライのノードを作成し、
トライ中のキー固有のコード列を表す固有コード列を作成し、
入力キーを１単位コードずつ取り出し、各単位コードを

トライの内部コードに変換し、　単位コードがノードに含まれるか否かを前記ラベルを用いて確認し、単位コードがノードに含まれることを確認した場合に、前記トライのノードから次のトライのノード情報と前記固有コード列へのポインタ情報のうちの１つを得て、次の単位コードに対する照合処理に進むことを特徴とする検索方法。

【請求項１１】　請求項１０記載の検索方法において、
複数のキーの集合を表す２進木を作成し、
該２進木を構成する複数のノード列のラベルを、互いに重複しないようにラベル配列に格納し、
該ラベル配列に対応して、前記次のトライの情報と前記固有コード列へのポインタ情報のうちの１つをトライ配列に格納し、
前記ラベル配列とトライ配列から前記圧縮されたトライのノードを作成することを特徴とする検索方法。

【請求項１２】　請求項１１記載の検索方法において、
前記ラベル配列内の第１の位置に前記２進木の第１のノード列のラベルを格納し、
前記圧縮されたトライのノード内で前記２進木の第２のノード列が挿入可能な第２の位置を求め、
該第２の位置に該第２のノード列を挿入し、
前記トライ配列内の前記第１の位置に、前記第２の位置を指す位置情報を格納することを特徴とする検索方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は検索装置および辞書／テキスト検索方法、特に高速かつ記憶容量の小さい検索装置およびそれを用いた辞書検索方法、テキスト検索方法に関する。

【０００２】機械翻訳、自然言語解析または音声認識等に用いる各種の辞書や、データベースをテキストのキーワードで検索する装置では、検索速度が高速で、かつインデックスサイズが小さいことが要求されている。

【０００３】

【従来の技術】従来の検索装置としては、ハッシュ法、Ｂ－Ｔｒｅｅ法、トライ（ＴＲＩＥ：tree retrieval）法などに基づく検索方式が提案され、利用されている。

【０００４】従来のトライ法を用いた検索装置の基本概念について説明する。図２１は、従来のトライ辞書検索装置の例を示している。図２１において、７０はトライノード選択機構、７１はトライノード格納装置を表す。

【０００５】例えば英文字見出しを持つ辞書検索では、１つのトライノードは終了記号（例えば＃）を含めた（２６＋１）個のトライノード要素からなり、辞書検索装置はこれらを格納するトライノード格納装置７１と、トライノード選択機構７０とからなる。

【０００６】図２１のトライノード格納装置７１では、ａ＃、ａｂｌｅ＃、ａｇｒｅｅ＃（＃はキーの終了記号）をキーとするトライノードの構成例を示している。最初のトライノードＮ１は、１文字目のキー情報を表しており、文字“ａ”が有効で、その位置に次のトライノードＮ２へのポインタが格納されている。２段目のトライノードＮ２では、２文字目の“＃”と“ｂ”と“ｇ”の文字が有効であり、“＃”の位置から検索目的のデータまたはそのデータへのアクセス情報がポイントされている。そして、トライノードＮ２の“ｂ”の位置から次のトライノードＮ３を、“ｇ”の位置からトライノードＮ６をポイントする。以下同様に、有効な文字についてのトライノードの連鎖が続く。

【０００７】トライノード選択機構７０は、入力キーに対して１文字ずつ取り出し、トライノード格納装置７１における最初のトライノードＮ１から順番に、該当する文字位置が有効なポインタ情報を保持しているかを確認し、トライノードの連鎖をたどって目的とするデータへのアクセス情報を得る。

【０００８】トライ法では、以上のようにキーの検索を先頭から１単位ずつトライノードと比較して行う。トライノード要素と文字との照合は、トライノードの文字番目の照合で行うことができるため、検索速度が高速である。

【０００９】

【発明が解決しようとする課題】従来の検索装置で用いられているハッシュ法には、キーの衝突が多く生じるような場合には制御が煩雑であり、オーバーヘッドが生じて検索速度が遅くなるか、またはキーの衝突を回避するために予め十分な大きさの記憶領域を用意しなければならないという、キーの衝突と記憶容量の問題があった。

【００１０】また、Ｂ－Ｔｒｅｅ法では、記憶容量の問題はないが、検索速度の高速性に欠けるという問題があった。一方、従来のトライ法は、検索速度は高速であるが、多くのトライノードが必要になるとともにトライノード内に無駄な領域が多いため、多大な記憶容量を必要とするという問題があり、キー数が多い場合などにはすべてのキーに対してトライ法で辞書を構成することは実際上不可能であるという問題があった。

【００１１】本発明は上記問題点の解決を図り、トライ法を用いる検索装置において検索の高速性を保ったままで、記憶容量の削減を可能とすることを目的とする。

【００１２】

【課題を解決するための手段】図１は本発明の構成例を示す。図１において、１０は検索処理を実行するＣＰＵやメモリなどからなる処理装置、１１は検索キーを入力するキー入力部、１２は入力キーの単位コードをトライの内部コードに変換するコード変換処理部、１３はトライノードを選択して検索を進めるノード選択処理部、１４はあらかじめ入力キーの単位コードとトライの内部コードとの対応情報を記憶するコード変換テーブル、２０は検索キーとデータとの対応情報を管理するインデックス格納装置、２１は圧縮されたトライのノードを格納す

るノード格納部、２２は圧縮されたトライのノードの照合の際に検索対象となる単位コードがノードに含まれるか否かを判定するためのトライのノードに対するラベルを格納するノード要素の確認用ラベル格納部、２３はトライ中のキー固有のコード列を格納する固有コード列格納部、３０は検索目的のデータを格納するデータ本体格納装置を表す。

【００１３】キー入力部１１は、検索対象となるキーを受理し、その入力キーをコード変換処理部１２に渡す。コード変換処理部１２は、入力キーを１単位コードずつ取り出し、所定の計算により、またはコード変換テーブル１４を用いることにより、各単位コードをトライの内部コードに変換し、ノード選択処理部１３に渡す。

【００１４】ノード格納部２１には、従来のトライノードを重ね合わせるようにして圧縮したトライのノードが格納されている。また、確認用ラベル格納部２２には、現在検索している単位コードがノード格納部２１のノード上にあるかどうかの確認用ラベルが格納されている。また、固有コード列格納部２３には、トライ中のキー固有のコード列が格納されている。　ノード選択処理部１３は、ノード格納部２１および確認用ラベル格納部２２を参照し、現在入力されている単位コードがトライノード中にあるかどうかを判断し、単位コードがあれば次のトライノードに移り、コード変換処理部１２から次の単位コードを受け取る。トライノード中に単位コードがなければ入力キーがデータ中にないと判断する。また、着目しているトライノードに固有コード列格納部２３におけるキーの固有コード列へのポインタが格納されている場合には、入力キーの残りのコード列またはその内部コード列を固有コード列格納部２３内のキーの固有コード列と照合する。コード列が一致してキーの照合ができれば、入力キーに対応するデータ本体格納装置３０中のデータへのアクセス情報を得る。そうでなければ入力キーに対応するデータは、データ本体格納装置３０中に存在しないと判断する。

【００１５】本発明では、従来の個々のトライノードを有効な要素が重複しないように重ね合わせて、圧縮したトライノードを構成する。これに伴い、現在検索している単位コードがノード上にあるかどうかを確認できるように、各ノード要素に対応して確認用のラベルを格納する。また、入力キーにおけるそのキーに固有の部分は、その部分だけを取り出し、ノード格納部２１とは別に設けた固有コード列格納部２３に格納する。

【００１６】こうすることにより、検索速度は従来とほぼ同様に維持したまま記憶容量を削減することができる。図１に示す検索装置と自然言語の単語とを組み合わせることにより圧縮したトライを構成し、辞書の見出しを管理することによって、検索速度が高速でインデックスサイズが小さい仮名漢字変換辞書、形態素解析装置、構文解析装置などを実現することができる。

【００１７】また、図１に示す検索装置と自然言語の音声データとを組み合わせることにより圧縮したトライを構成し、音声データに対応する認識カテゴリを管理することによって、検索速度が高速でインデックスサイズが小さい音声認識用の辞書を実現することができる。

【００１８】さらにまた、図１に示す検索装置とテキストデータとを組み合わせることにより圧縮したトライを構成し、データベースへのインデックスとすることによって、検索速度が高速でインデックスサイズが小さいデータベースのテキスト検索装置を実現することができる。

【００１９】

【発明の実施の形態】図２は本発明の実施形態におけるコード変換テーブルとインデックス格納装置の構成例を示す図である。

【００２０】例えば英文字見出しを持つ辞書検索装置の場合、図２（Ａ）に示すように、キーの終了を示す記号＃と英文字ａ～ｚの文字コードとを、１～２７のトライの内部コードに変換するための情報を持つコード変換テーブル１４を用意する。なお、コード変換処理部１２では、コード変換テーブル１４を用いないで、文字コードを２進数の数値とみた算術演算により、入力キーの単位コードをトライの内部コードに変換するようにしてもよい。

【００２１】図２（Ｂ）は、キーとして“ａ＃”、“ａｂｎｏｍａｌ＃”、“ａｇｒｅｅ＃”、“ｂａｃｈｅｌｏｒ＃”、“ｂｃｓ＃”の５個のキーが存在する場合のノード格納部２１（ＴＲＩＥと表す）と、確認用ラベル格納部２２（ＣＨＥＣＫと表す）における格納状態を示している。

【００２２】“ａ＃”、“ａｂｎｏｍａｌ＃”、“ａｇｒｅｅ＃”、“ｂａｃｈｅｌｏｒ＃”、“ｂｃｓ＃”の各キーの中で他のキーとの区別に影響のない部分については、その部分を該当するキーの固有コード列として抽出し、図２（Ｃ）に示すように、固有コード列格納部２３に格納する。

【００２３】図２に示す辞書の作成方法について、図３に従って説明する。上述のようにキーとしては、図３（Ａ）に示す５個のキーがあるものとする。説明を簡単にするために、これらのキーの各文字を図３（Ｂ）に示すように木構造に展開した場合を想定する。この木構造で、各キーの枝分かれのない文字列部分を抽出することにより、図３（Ｃ）に示すような固有コード列格納部２３に格納すべきキーの固有コード列が得られる。なお、固有コード列格納部２３における各コード列は、入力キーの元の文字コード列であっても、それをトライの内部コードに変換した後の内部コード列であっても、どちらでもよい。

【００２４】ノード格納部２１と確認用ラベル格納部２２は、次のように作成する。まず、図３（Ｂ）に示す木

構造における従来のトライノードに相当するトライノードＮ１、Ｎ２、Ｎ３の部分に着目する。ノード要素の無効な部分を“０”で表し、有効な部分をその文字記号で表すと、図３（Ｄ）に示すようなトライノードＮ１～Ｎ３になる。これらを有効な部分が重複しないようにずらして、１本のトライノードに重ね合わせる。これをもとに、図３（Ｄ）に示すノード格納部２１を作成する。なお、このノード格納部２１のトライノードにおける先頭には処理の便宜上、“１”のインデックス値を持つノード要素を付加する。

【００２５】例えばノード格納部２１の３番目のエントリ（最初の“ａ”の部分）には、次のトライノードＮ２に相当するノード列が、トライノードＮ１の４番目（ノード格納部２１の５番目）のエントリから始まるので、インデックス値として“４”を格納する。また、ノード格納部２１の４番目のエントリ（“ｂ”の部分）には、次のトライノードＮ３に相当するノード列が、トライノードＮ１の６番目（ノード格納部２１の７番目）のエントリから始まるので、インデックス値として“６”を格納する。

【００２６】また、ノード格納部２１における各ノード要素に対応する単位コードの後に続くコードが、固有コード列のみからなるとき、そのノード要素に固有コード列格納部２３における固有ノード列へのポインタＫ１～Ｋ５を格納する。なお、ノード要素がトライノードへのポインタであるか固有コード列へのポインタであるかは、例えば先頭ビットをフラグとして用いて区別することができる。より具体的には、トライノードへのポインタとして正の値を用い、固有ノード列へのポインタとして負の値を用いることが考えられる。

【００２７】確認用ラベル格納部２２の各エントリには、ノード格納部２１における各ノード要素に対応するキーの文字コードまたは内部コードを確認用ラベルとして格納する。ノード格納部２１および確認用ラベル格納部２２における無効のエントリにはその旨を示すｎｕｌｌコード（“０”）などを設定する。

【００２８】以上のようにすることによって、図２（Ｂ）、（Ｃ）に示すようなＴＲＩＥ、ＣＨＥＣＫおよび固有コード列を持つインデックス格納装置２０を作成することができる。なお、ここではＴＲＩＥは、圧縮したトライのノードを格納した整数配列であり、ＣＨＥＣＫは、文字がトライのノードに含まれるか否かを確認するための文字ラベルを格納した要素確認用の各１バイトの文字配列である。

【００２９】従来のトライで構成すると、トライノードの領域として１２８×２２要素の領域が必要であるのに対し、本実施形態では、図２から明らかなように１２要素＋１２文字＋２２文字分の領域で済んでいる。

【００３０】図４は、本発明の実施形態による検索処理のフローチャートである。まず、ステップＳ１では、入力装置または検索要求プログラム等から検索のキーを入力する。ステップＳ２では、トライ検索のためのポインタをノード格納部２１および確認用ラベル格納部２２における最初のノードにセットし、以下、ステップＳ３以降を実行する。

【００３１】ステップＳ３では、入力キーを比較するために先頭から１単位コードずつ要素を取り出す。例えば入力キーが文字列で構成される場合には、文字コードを１文字ずつ取り出すことになる。ステップＳ４では、取り出した単位コードをコード変換テーブル１４を用いて、または算術演算もしくはビット操作により、トライの内部コードに変換する。

【００３２】次にステップＳ５により、トライ検索のためのポインタを内部コード分進める。そして、ステップＳ６により、確認用ラベル格納部２２を参照し、ポインタの指す確認用のラベルが現在のコードと一致するかどうかを判定する。一致しない場合には、ステップＳ９により検索失敗として検索要求元へ通知する。

【００３３】確認用のラベルが現在のコードと等しい場合、ステップＳ７によってノード格納部２１におけるポインタの示す要素が辞書中のキーの固有コード列格納部２３を指しているかどうかを判定する。固有コード列格納部２３を指していない場合、ステップＳ８によって、ノード格納部２１のノード要素から得た値をポインタの値として、次のトライのノードにポインタを進める。その後、ステップＳ３に戻り、次の単位コードについて同様に処理を繰り返す。

【００３４】ステップＳ７において、ポインタの先が辞書中の固有コード列格納部２３を指していると判定された場合には、ステップＳ１０へ進み、キーの残りとポイントされた固有コード列とを照合する。キーの残りと固有コード列とが等しい場合、ステップＳ１１によって検索成功とし、入力キーに対するデータへのアクセス情報を得て、検索要求元へ通知する。一方、キーの残りと固有コード列とが等しくない場合、ステップＳ１２により検索失敗として検索要求元へ通知する。

【００３５】次に、図４に示す処理の流れに従って、図２に示すインデックス格納装置を用いた場合の具体的な検索例について説明する。

〔キー“ａ＃”の検索要求があった場合（検索の成功例）〕

（１）キー“ａ＃”を入力する（図４のＳ１）。

（２）ポインタの値を最初のノードを示すように“１”とする（Ｓ２）。

（３）入力キー“ａ＃”から文字コード“ａ”を取り出す（Ｓ３）。

（４）文字コード“ａ”をトライの内部コード“２”に変換する（Ｓ４）。

（５）ポインタ“１”に内部コード“２”を加算し、ポインタの値を“３”に進める（Ｓ５）。

（６）ポインタの値が“３”であることから、ＣＨＥＣＫの第３要素に格納されている確認用ラベル“ａ”と現在のコード“ａ”とが等しいかどうかを判定する（Ｓ６）。
（７）ここでは等しいので、次にＴＲＩＥの第３要素が固有コード列を指しているかどうかを判定する（Ｓ７）。
（８）ＴＲＩＥの第３要素は“４”であり、固有コード列を指していないので、ポインタの値を“４”にする（Ｓ８）。
（９）入力キー“ａ＃”から次の文字コード“＃”を取り出す（Ｓ３）。
（１０）文字コード“＃”をトライの内部コード“１”に変換する（Ｓ４）。
（１１）ポインタ“４”に内部コード“１”を加算し、ポインタの値を“５”に進める（Ｓ５）。
（１２）ポインタの値が“５”であることから、ＣＨＥＣＫの第５要素に格納された確認用ラベル“＃”と現在のコード“＃”とが等しいかどうかを判定する（Ｓ６）。
（１３）ここでは等しいので、次にＴＲＩＥの第５要素が固有コード列を指しているかどうかを判定する（Ｓ７）。
（１４）ＴＲＩＥの第５要素は固有コード列格納部２３の“Ｋ１”を指しているので、キーの残りと“Ｋ１”の固有コード列の内容とが等しいかを判定する（Ｓ１０）。
（１５）キーの残りはなく、また“Ｋ１”の固有コード列の内容も空（φ）であるので、等しいと判定され、検索成功となる（Ｓ１１）。
【００３６】〔キー“ａｃ＃”の検索要求があった場合（検索の失敗例）〕
（１）キー“ａｃ＃”を入力する（図４のＳ１）。
（２）ポインタの値を最初のノードを示すように“１”とする（Ｓ２）。
【００３７】以下、（３）から（８）までの処理は上述のキー“ａ＃”の検索と同様である。
（９）入力キー“ａｃ＃”から次の文字コード“ｃ”を取り出す（Ｓ３）。
（１０）文字コード“ｃ”をトライの内部コード“４”に変換する（Ｓ４）。
（１１）ポインタ“４”に内部コード“４”を加算し、ポインタの値を“８”に進める（Ｓ５）。
（１２）ポインタの値が“８”であることから、ＣＨＥＣＫの第８要素に格納された確認用ラベル“ａ”と現在のコード“ｃ”とが等しいかどうかを判定する（Ｓ６）。
（１３）確認用ラベル“ａ”と現在のコード“ｃ”とは等しくないので、キー“ａｃ＃”はないことが分かり、検索失敗となる（Ｓ９）。
【００３８】図５は、本発明の実施形態による日本語の単語をキーとする辞書検索装置のコード変換テーブルとインデックス格納装置の構成例を示す図である。図５の例では、“あ＃”、“あさがお＃”、“いろ＃”、“居る＃”、“居住＃”、“嘘＃”、“嘘つき＃”の日本語文字列をキーとする辞書を構成している。
【００３９】コード変換テーブル１４は、図５（Ａ）に示すように構成され、ノード格納部２１および確認用ラベル格納部２２は、図５（Ｂ）に示すように構成され、固有コード列格納部２３は、図５（Ｃ）に示すように構成される。これを従来のトライで構成すると約６０００×１３要素の領域が必要であるのに対し、本実施形態では１５要素＋１５文字＋１１文字分の領域で済んでいる。
【００４０】図６は、図５に示す辞書の作成方法を示している。図６（Ａ）は、上述の７つの日本語のキーを示している。これらのキーは、図７（Ｂ）に示すような木構造に展開される。図７（Ｂ）の木構造において、トライノードＮ１、Ｎ２、Ｎ３、Ｎ４に含まれない文字列部分を抽出することにより、図７（Ｃ）に示すような固有コード列が得られる。
【００４１】以下に、この辞書の検索例を説明する。
〔キー“あさがお＃”の検索要求があった場合（検索の成功例）〕
（１）キー“あさがお＃”を入力する（図４のＳ１）。
（２）ポインタの値を最初のノードを示すように“１”とする（Ｓ２）。
（３）入力キー“あさがお＃”から文字コード“あ”を取り出す（Ｓ３）。
（４）文字コード“あ”をトライの内部コード“２”に変換する（Ｓ４）。
（５）ポインタ“１”に内部コード“２”を加算し、ポインタの値を“３”に進める（Ｓ５）。
（６）ポインタの値が“３”であることから、ＣＨＥＣＫの第３要素に格納された確認用ラベル“あ”と現在のコード“あ”とが等しいかどうかを判定する（Ｓ６）。
（７）ここでは等しいので、次にＴＲＩＥの第３要素が固有コード列を指しているかどうかを判定する（Ｓ７）。
（８）ＴＲＩＥの第３要素は“４”であり、固有コード列を指していないので、ポインタの値を“４”にする（Ｓ８）。
（９）入力キーの“あさがお＃”から次の文字コード“さ”を取り出す（Ｓ３）。
（１０）文字コード“さ”をトライの内部コード“４”に変換する（Ｓ４）。
（１１）ポインタ“４”に内部コード“４”を加算し、ポインタの値を“８”に進める（Ｓ５）。
（１２）ポインタの値が“８”であることから、ＣＨＥＣＫの第８要素に格納された確認用ラベル“さ”と現在

のコード“さ”とが等しいかどうかを判定する（Ｓ６）。
（１３）ここでは等しいので、次にＴＲＩＥの第８要素が固有コード列を指しているかどうかを判定する（Ｓ７）。
（１４）ＴＲＩＥの第８要素は固有コード列格納部２３の“Ｋ３”を指しているので、キーの残りと“Ｋ３”の固有コード列の内容とが等しいかを判定する（Ｓ１０）。
（１５）キーの残りは“がお＃”であり、また“Ｋ３”の位置の固有コード列も“がお＃”であるので、等しいと判定され、検索成功となる（Ｓ１１）。
【００４２】〔キー“あさ＃”の検索要求があった場合（検索の失敗例）〕
（１）キー“あさ＃”を入力する（図４のＳ１）。
（２）ポインタの値を最初のノードを示すように“１”とする（Ｓ２）。
【００４３】以下、（３）から（１３）までの処理は上述のキー“あさがお＃”の検索と同様である。
（１４）ＴＲＩＥの第８要素は固有コード列格納部２３の“Ｋ３”を指しているので、キーの残りとその固有コード列“Ｋ３”とが等しいかを判定する（Ｓ１０）。
（１５）キーの残りは“＃”であり、また“Ｋ３”の位置の固有コード列は“がお＃”であるので、等しくないと判定され、検索失敗となる（Ｓ１２）。
【００４４】次に、本発明の圧縮されたトライの作成方法について詳細に説明する。図７は、トライノードを圧縮して配列ＴＲＩＥとＣＨＥＣＫを生成するトライ圧縮処理のフローを示している。
【００４５】まず、ステップＳ２１では、処理装置１０はトライの作成対象となるキー集合をソートする。次に、ステップＳ２２で、キー集合を構成する文字をカウントし、各文字に対して頻度の大きいものから順に、１，２，３，．．．，ｎという数字を内部コードとして割り振り、コード変換テーブル１４を作成する。そして、ステップＳ２３で、ソートされたキー集合の２進木を作成する。このとき、最小接辞部分から構成され、圧縮されたトライのノードへのポインタを持った２進木を作成する。
【００４６】図３（Ａ）に示すようなキー集合からは、例えば図８のような２進木が作成される。図８の最小接辞・データ領域には、図３（Ｃ）の固有コード列に対応する最小接辞部またはデータが格納されている。また、このときのコード変換テーブル１４は図２（Ａ）のようになる。
【００４７】次に、ステップＳ２４以降の処理により、２進木を圧縮されたトライへ変換する。ステップＳ２４では、配列ＴＲＩＥとＣＨＥＣＫを初期化し、ステップＳ２５で、２進木内の位置を示すポインタの初期状態として、それを２進木のルートノードにセットする。また、ステップＳ２６で、２進木のルートノードが挿入される配列内の位置を、ｉｎｄｅｘ＝１の位置に設定する。
【００４８】ここでは、配列ＴＲＩＥとＣＨＥＣＫの初期状態は図９のようになる。次に、ステップＳ２７で、現在のポインタが指す２進木のノード（ポインタノード）を先頭とするノード列（ノードリスト）が挿入可能な、ＴＲＩＥとＣＨＥＣＫ内の位置を求める。挿入しようとしているノードリスト中の各要素の位置に対応する配列の領域が使用されていなければ、そのときのノードリストの先頭に対応する位置が挿入可能位置となる。この挿入可能位置は、配列の先頭から最後尾に向かって順に捜していき、最初に見つかった位置とする。
【００４９】次に、ステップＳ２８で、ＴＲＩＥ内の２進木挿入位置に、ステップＳ２７で求めた挿入可能位置のｉｎｄｅｘ値を設定する。例えば、２進木のポインタノードが図８のルートノードを指す場合は、ルートノードが挿入される位置はステップＳ２６でｉｎｄｅｘ＝１に決まっている。このときステップＳ２８で、ｉｎｄｅｘ＝１の位置に、挿入するリストの開始位置の値を入れる必要があるが、ルートノードの場合には初期化処理により既に設定されているので、Ｓ２８の処理は省略される。
【００５０】次に、ステップＳ２９で、２進木のポインタノードより始まるリストに連なる各ノードのラベルを、ＣＨＥＣＫ内の対応する部分に書き込む。ここでは、リストに繋がっているラベルはａとｂであり、それらのトライの内部コードはそれぞれ２と３である。また、ノードの挿入可能位置はｉｎｄｅｘ＝１の位置であるので、ａの書込み位置は１＋２＝３となり、ｂの書込み位置は１＋３＝４となる。したがって、ステップＳ２９の処理の結果は図１０のようになる。
【００５１】また、ステップＳ３０で、２進木のポインタノードより始まるリストに連なる各ノードに、ステップＳ２９でセットしたＣＨＥＣＫに対応するＴＲＩＥへのポインタをセットする。そして、これらのポインタが指すＴＲＩＥ内の位置を２進木挿入位置とする。
【００５２】ここでは、ラベルａ、ｂに対応して、２進木のノードａ、ｂからＴＲＩＥへポインタが張られる。これらをそれぞれ@１ａ、@１ｂと表すと、ステップＳ３０の処理の結果は図１１のようになる。
【００５３】次に、ステップＳ３１で、２進木のポインタノードがデータを持っているかどうかを確認する。それがデータを持っていなければ、ステップＳ３２で、２進木のポインタに現在のノードの子ノードをセットして、ステップＳ２７以降の処理を再帰的に繰り返す。
【００５４】ステップＳ３１の条件判定で、２進木のポインタノード（ａ→ｂ→ＮＵＬＬのうちのａのノード）はデータを持っていないので、処理はステップＳ３２に移る。ステップＳ３２では、ポインタノードの子ノード

を２進木のポインタノードとしてセットする。この結果、ポインタノードは＃→ｂ→ｇ→ＮＵＬＬのうちの＃のノードとなる。このステップＳ３２からの処理は再帰処理となり、その終了後はステップＳ３４から処理が始まる。

【００５５】ステップＳ２７では、２進木の挿入可能位置を次のようにして求める。ポインタノードより始まる２進木のリスト＃→ｂ→ｇ→ＮＵＬＬの各ノードのラベルを、図２（Ａ）のコード変換テーブル１４を用いてトライの内部コードに対応させると、図１２のようになる。求める挿入可能位置は、上のラベルがＣＨＥＣＫ内のラベルと重複せず、かつ、トライのノードの先頭位置となっていない位置である。ここで、トライのノードの先頭位置とは、ｉｎｄｅｘ＝１のようにそこから始まるノードリストが存在する位置である。

【００５６】挿入可能位置を探索するには、原理的には、ｉｎｄｅｘの値が１の位置から順にラベルの重複がないかどうかをチェックし、候補となる位置がノードの先頭位置となっていないかどうかををチェックし、これらの条件に合う位置のｉｎｄｅｘの値を返せばよい。この方法による探索は、要素がまだ挿入されていない位置に到達した時点で必ず停止するので、その終了は保証されている。

【００５７】しかし、実際には計算時間が（ｉｎｄｅｘサイズ×挿入するノードの数）に比例し、ｉｎｄｅｘの数が大きくなるにつれ現実的な時間ではなくなる。そこで、配列がｉｎｄｅｘの値の小さい方から詰められ、かつ１度挿入されたノードの移動、削除がないことを利用して、配列ＴＲＩＥ、ＣＨＥＣＫの使用率をモニターしながら、探索を開始する位置を調整すればよい。言い換えれば、探索開始位置のｉｎｄｅｘの値を段々大きくしていけばよい。これにより、実際の探索範囲を小さくし、実用に供することができる。

【００５８】こうして、ここでは図１３のように、リスト＃→ｂ→ｇ→ＮＵＬＬの挿入可能位置が求められる。図１３より、ラベル＃の直前の位置が挿入可能位置となり、そのｉｎｄｅｘの値は４となる。

【００５９】ステップＳ２８では、ポインタ＠１ａが指す２進木挿入位置へステップＳ２７で得られた挿入可能位置のｉｎｄｅｘの値４を入れる。この結果、図１４のようになる。

【００６０】ステップＳ２９では、ステップＳ２７で得られた挿入可能位置から２進木のポインタノードより始まるリストのラベルをＣＨＥＣＫに書き込む。リストに繋がっているラベルは＃、ｂ、ｇであり、それらのトライの内部コードはそれぞれ１、３、８である。また、ノードの挿入可能位置はｉｎｄｅｘ＝４であるので、ステップＳ２９の結果は図１５のようになる。

【００６１】ステップＳ３０では、２進木から配列ＣＨＥＣＫに対応する配列ＴＲＩＥの位置へポインタを張る。これらを＠２＃、＠２ｂ、＠２ｇと表すと、ステップＳ３０の結果は図１６のようになる。

【００６２】ステップＳ３１の条件判定で、２進木のポインタノード（＃→ｂ→ｇ→ＮＵＬＬの＃のノード）はデータを持っているので、処理はステップＳ３３に移る。ステップＳ３３では、２進木のポインタノードが指すデータへのポインタを、ステップＳ３０でセットしたポインタが指すＴＲＩＥの要素にセットする。

【００６３】ここでは、２進木の＃のノードから最小接辞部へのポインタを、対応するＴＲＩＥの要素にセットする。この場合、最小接辞部の要素は空である。最小接辞・データ領域内の要素（データもしくは最小接辞部）へのポインタをＫ１と表すと、ステップＳ３３の結果は図１７のようになる。図１７では、Ｋ１＝空である。

【００６４】次に、ステップＳ３４では、２進木のポインタノードを同じリスト内で１つ進め、次の要素にセットする。ここでは、ポインタノードはリスト＃→ｂ→ｇ→ＮＵＬＬのうちのｂのノードとなる。　次に、ステップＳ３５で、２進木のポインタノードがデータを持っているかどうかをチェックする。データがあれば処理はステップＳ３３に移り、なければステップＳ３６に移る。ステップＳ３６では、ポインタがＮＵＬＬかどうかをチェックし、ＮＵＬＬであれば処理を終了し、ＮＵＬＬでなければ処理はステップＳ３７に移る。

【００６５】ステップＳ３７では、２進木のポインタに現在のノードの子ノードをセットして、ステップＳ２７以降の処理を再帰的に繰り返す。ステップＳ３５の条件判定で、２進木のポインタノード（＃→ｂ→ｇ→ＮＵＬＬのｂのうちのノード）はデータを持っているので、処理はステップＳ３３に移る。以下、ｇのノードまで同様の処理が続く。その結果、図１８に示すように、Ｋ１＝空へのポインタ、Ｋ２＝ｎｏｒｍａｌ＃へのポインタ、Ｋ３＝ｒｅｅ＃へのポインタがそれぞれＴＲＩＥに格納される。ここで、２進木ポインタがＮＵＬＬとなり処理は一旦終了するが、再帰処理なので呼び出し元にもどり、ステップＳ３４から処理が再び始まる。この場合には、２進木のポインタノードはａ→ｂ→ＮＵＬＬのうちのｂのノードとなる。

【００６６】その後、ステップＳ３５、Ｓ３６の処理が行われ、ｂのノードの子ノードに対してステップＳ２７からの一連の再帰処理が行われる。その結果、配列ＴＲＩＥ、ＣＨＥＣＫは図１９のようになる。図１９において、Ｋ１＝空へのポインタ、Ｋ２＝ｎｏｒｍａｌ＃へのポインタ、Ｋ３＝ｒｅｅ＃へのポインタ、Ｋ４＝ｃｈｅｌｏｒ＃へのポインタ、Ｋ５＝ｓ＃へのポインタを表す。

【００６７】この処理が終った段階で、２進木のポインタノードはａ→ｂ→ＮＵＬＬのＮＵＬＬのノードとなり、ステップＳ３６でポインタが空の条件に適合するため、処理は終了する。この場合には、対応する再帰処理

の呼び出し元がないので、全処理が完了することになる。こうして生成された図１９のＴＲＩＥ、ＣＨＥＣＫは、図２（Ｂ）のＴＲＩＥ、ＣＨＥＣＫと一致している。

【００６８】このように、処理装置１０は２進木のルートから処理を開始して、２進木のルートの圧縮トライ配列への挿入場所の確保、２進木の各ノードから圧縮トライ配列へのポインタのセットの順に処理を進める。そして、ノードにデータがなければ、現在のポインタノードの子ノードを新たにポインタノードとして、再帰的に処理を行う。それ以外の場合には、２進木の各ノードから圧縮トライ配列へのポインタに対して、データへのポインタをセットし、ポインタノードをリスト上で次のノードに進める。２進木のリストの終端ノードは空であるため、２進木のルートノードからのリストの最後の処理が終了した時点で、トライ圧縮処理は終了する。

【００６９】図２０は、本発明の適用例を示す図である。図２０（Ａ）の例では、自然言語の単語４０を入力キーとし、本発明を用いた辞書検索装置４１により、辞書４２の検索を行うようにしている。例えば、ワードプロセッサの仮名漢字変換辞書、自然言語解析装置や機械翻訳装置における形態素解析または構文解析に用いる辞書への応用が可能である。

【００７０】図２０（Ｂ）の例では、マイクロフォンなどからの音声入力５０の信号をＡ／Ｄ変換装置５１によりアナログ／ディジタル変換し、それからデータ抽出装置５２によって抽出した音声特徴パラメータなどの音声データを入力キーとして、本発明を用いた辞書検索装置５３により辞書５４の検索を行うようにしている。本発明の技術と音声データとを組み合わせることにより圧縮したトライを構成し、効率的な音声認識等のための辞書の検索を実現することができる。

【００７１】図２０（Ｃ）の例では、入力キーとしてテキスト６０を入力し、本発明を用いたテキスト検索装置６１によりデータベース６２を検索できるようにしている。これにより、テキストをキーとする日本語文書などのデータベース６２の検索を、効率的に行うことが可能になる。

【００７２】

【発明の効果】以上説明したように、本発明によれば、日本語文字でも、英語文字でも、速度および記憶容量の両側面において電子化辞書の検索を効率よく行うことができるようになる。また、このような辞書システムを用いることにより、ワードプロセッサの仮名漢字変換辞書、形態素解析装置、構文解析装置などの基本機能を、より効率化することが可能となる。

【図面の簡単な説明】

【図１】本発明の構成例を示す図である。

【図２】本発明の実施形態におけるコード変換テーブルとインデックス格納装置の構成例を示す図である。

【図３】図２に示す辞書の作成方法説明図である。

【図４】本発明の実施形態による検索処理のフローチャートである。

【図５】本発明の実施形態による日本語の単語をキーとする辞書検索装置のコード変換テーブルとインデックス格納装置の構成例を示す図である。

【図６】図５に示す辞書の作成方法説明図である。

【図７】トライ圧縮処理のフローチャートである。

【図８】英語の２進木の例を示す図である。

【図９】トライノード圧縮時の配列データの例を示す図（その１）である。

【図１０】トライノード圧縮時の配列データの例を示す図（その２）である。

【図１１】トライノード圧縮時の配列データの例を示す図（その３）である。

【図１２】ラベルから内部コードへの変換例を示す図である。

【図１３】トライノード圧縮時の配列データの例を示す図（その４）である。

【図１４】トライノード圧縮時の配列データの例を示す図（その５）である。

【図１５】トライノード圧縮時の配列データの例を示す図（その６）である。

【図１６】トライノード圧縮時の配列データの例を示す図（その７）である。

【図１７】トライノード圧縮時の配列データの例を示す図（その８）である。

【図１８】トライノード圧縮時の配列データの例を示す図（その９）である。

【図１９】トライノード圧縮時の配列データの例を示す図（その１０）である。

【図２０】本発明の適用例を示す図である。

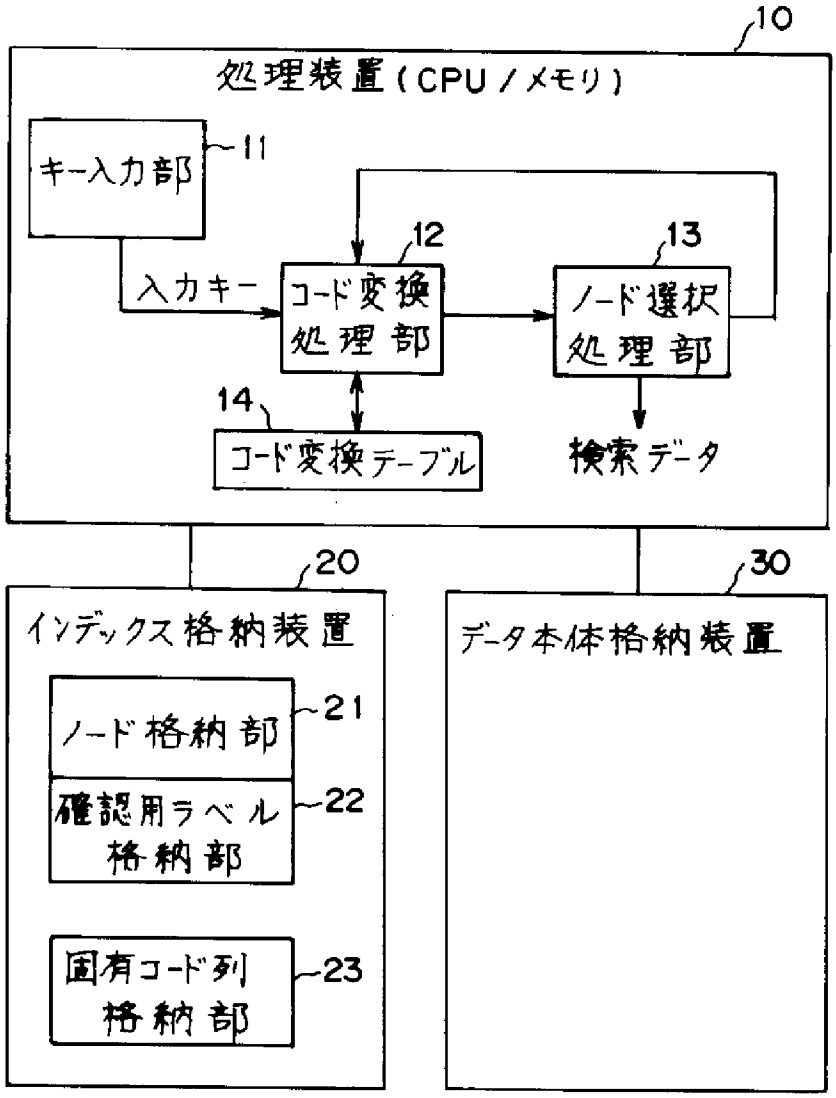
【図２１】従来のトライ辞書検索装置の例を示す図である。

【符号の説明】

１０　処理装置
１１　キー入力部
１２　コード変換処理部
１３　ノード選択処理部
１４　コード変換テーブル
２０　インデックス格納装置
２１　ノード格納部
２２　確認用ラベル格納部
２３　固有コード列格納部
３０　データ本体格納装置

【図１】

本発明の構成例を示す図



【図１２】

ラベルから内部コードへの変換例を示す図

#	b		g
1	0	3	0	0	0	0	8

【図１４】

トライノード圧縮時の配列データの例を示す図
（その５）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	@1b	0	0	0	0	0	0	0	0
CHECK	0	0	a	b	0	0	0	0	0	0	0	0

【図２】

コード変換テーブルとインデックス格納装置の構成例

(A) コード変換テーブル

14

文字コード	#	a	b	c	d	e	f	g	…	z
トライの内部コード	1	2	3	4	5	6	7	8	…	27

(B) 圧縮されたトライと要素確認用の配列

21	index	1	2	3	4	5	6	7	8	9	10	11	12
	TRIE	1	0	4	6	K1	0	K2	K4	0	K5	0	K3
22	CHECK	0	0	a	b	#	0	b	a	0	c	0	g

(C) キーの固有コード列

23

	K1	K2	K3	K4	K5
固有コード列	φ	normal#	ree #	chelor#	s #

【図５】

コード変換テーブルとインデックス格納装置の構成例

(A) コード変換テーブル

14

文字コード	#	あ	い	さ	る	つ	住	居	嘘	…
トライの内部コード	1	2	3	4	5	6	7	8	9	…

(B) 圧縮されたトライと要素確認用の配列

21

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRIE	1	0	4	K1	K2	K6	0	K3	8	5	K7	0	K4	0	K5
CHECK	0	0	あ	い	#	#	0	さ	居	嘘	つ	0	る	0	住

22

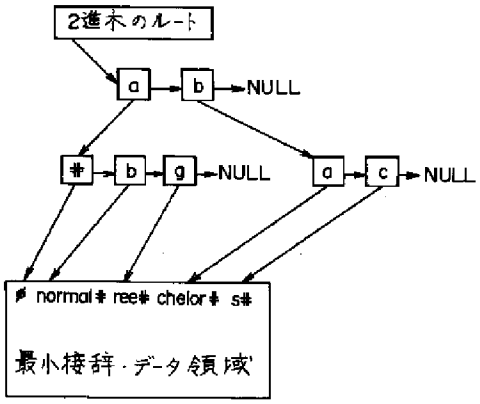
(C) キーの固有コード列

23

	K1	K2	K3	K4	K5	K6	K7
固有コード列	る#	φ	がお#	#	#	φ	き#

【図８】

英語の2進木の例を示す図



【図９】

トライノード圧縮時の配列データの例を示す図
（その1）

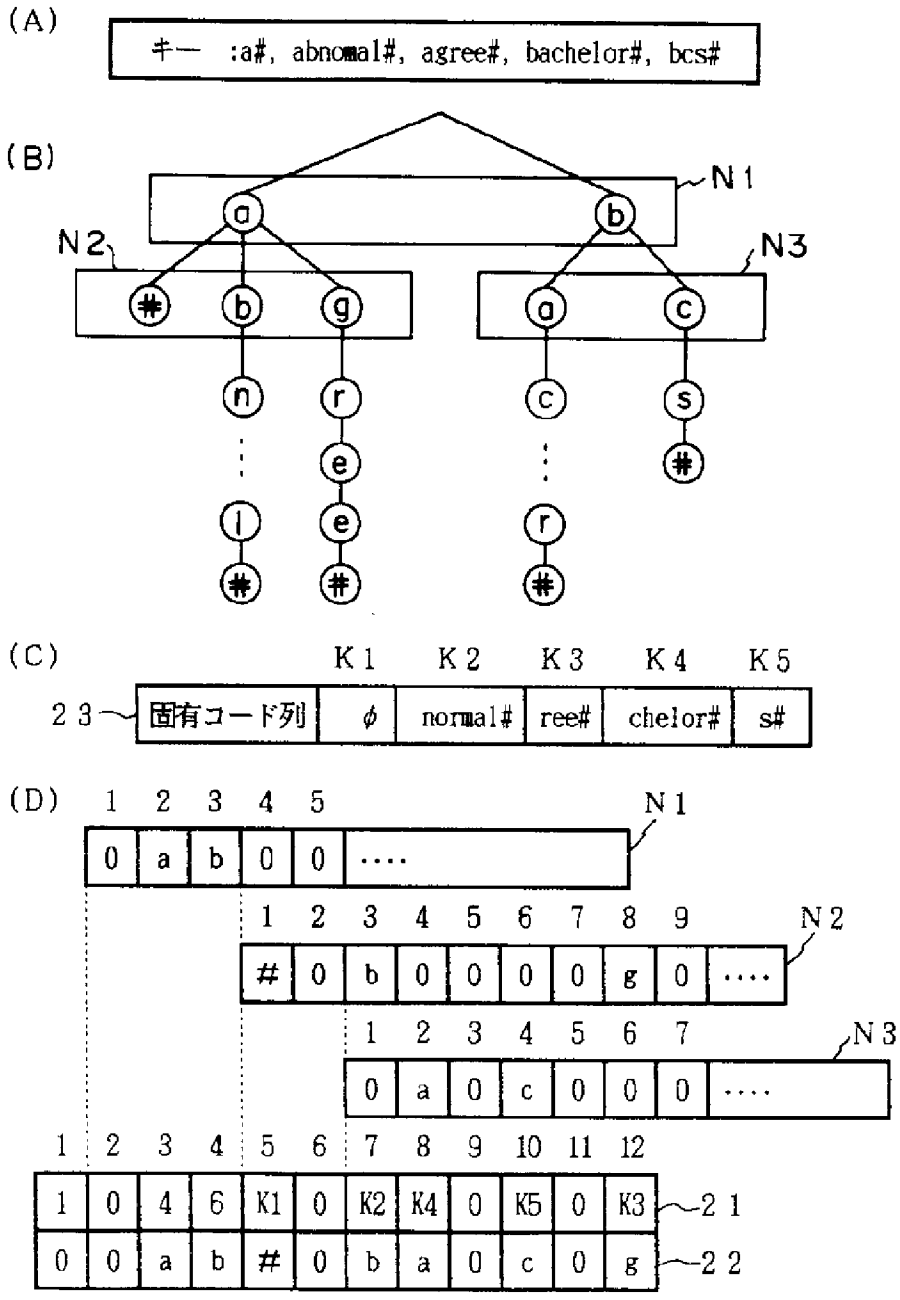
INDEX	1	2	3	4	5	6	7	8
TRIE	1	0	0	0	0	0	0	0
CHECK	0	0	0	0	0	0	0	0

【図１０】

トライノード圧縮時の配列データの例を示す図
（その2）

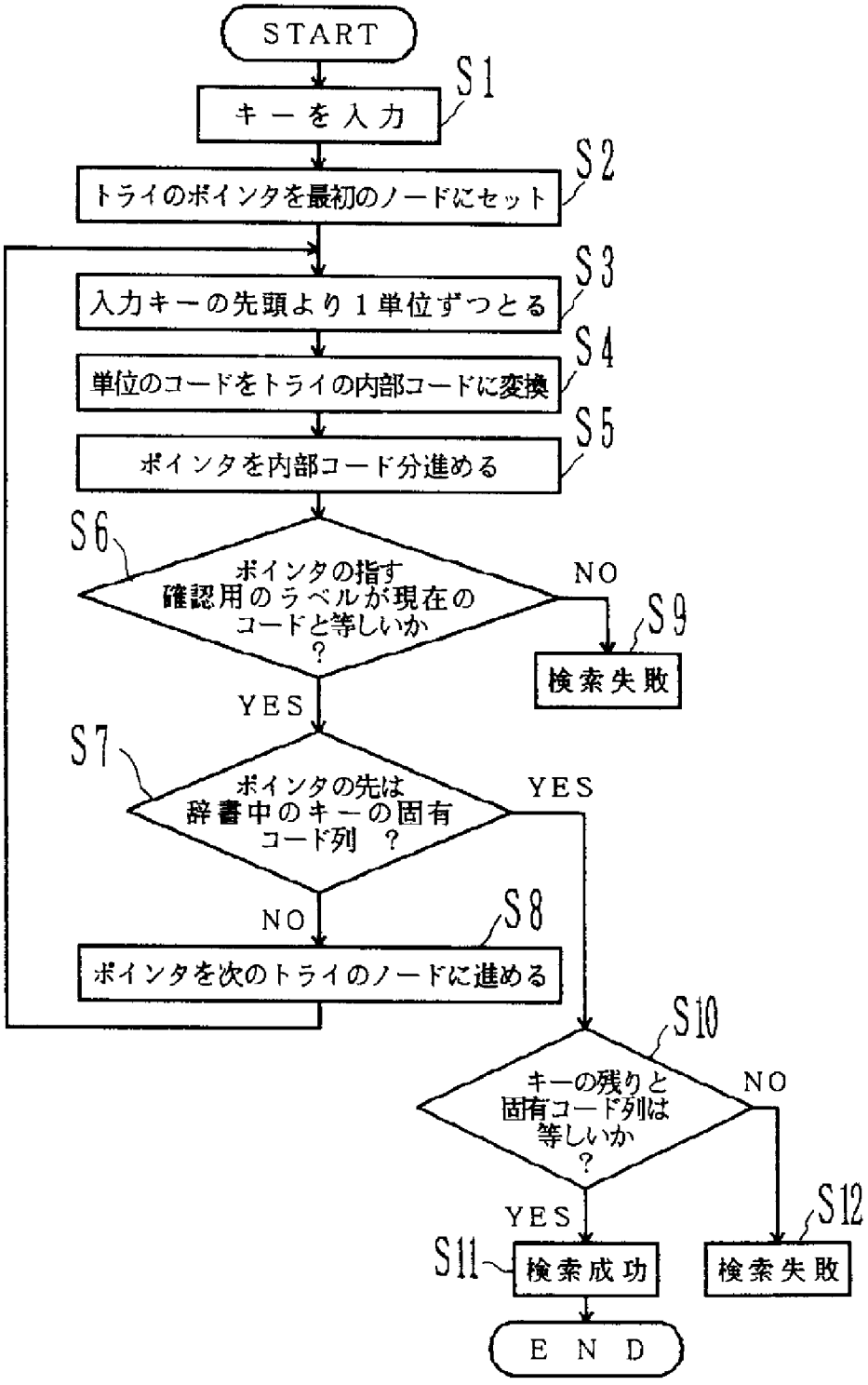
INDEX	1	2	3	4	5	6	7	8
TRIE	1	0	0	0	0	0	0	0
CHECK	0	0	a	b	0	0	0	0

図２に示す辞書の作成方法を説明する図



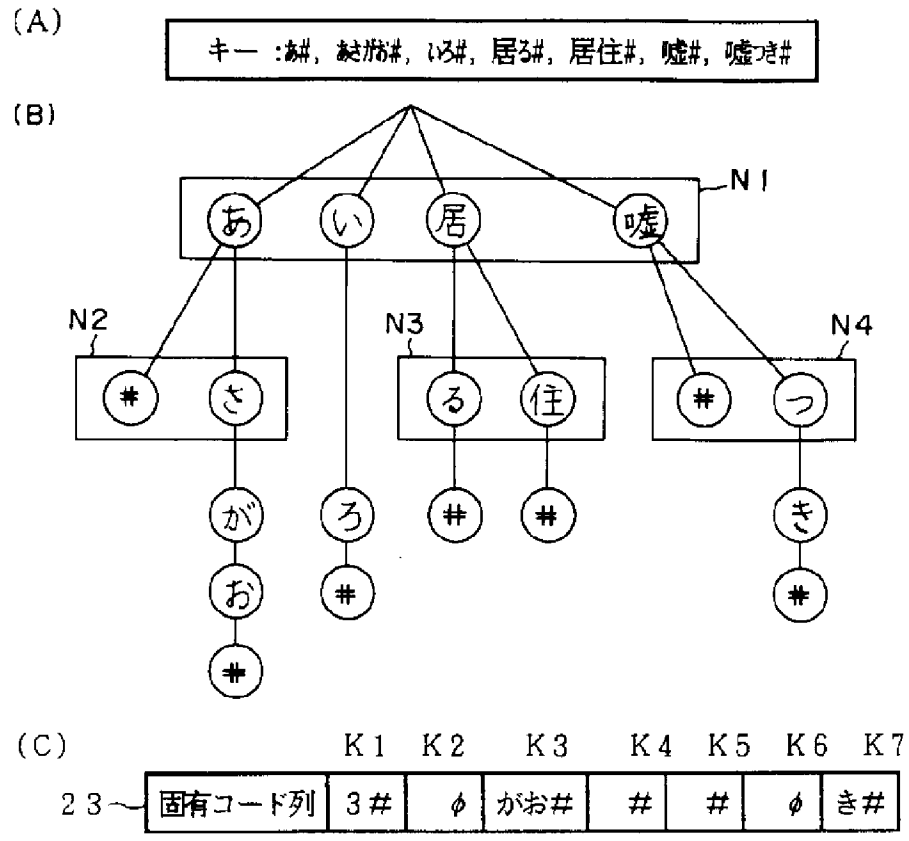
【図４】

本発明の実施形態による検索処理のフローチャート



【図６】

図５に示す辞書の作成方法説明図



【図１１】

トライノード圧縮時の配列データの例を示す図
（その３）

INDEX	1	2	3	4	5	6	7	8
TRIE	1	0	@1a	@1b	0	0	0	0
CHECK	0	0	a	b	0	0	0	0

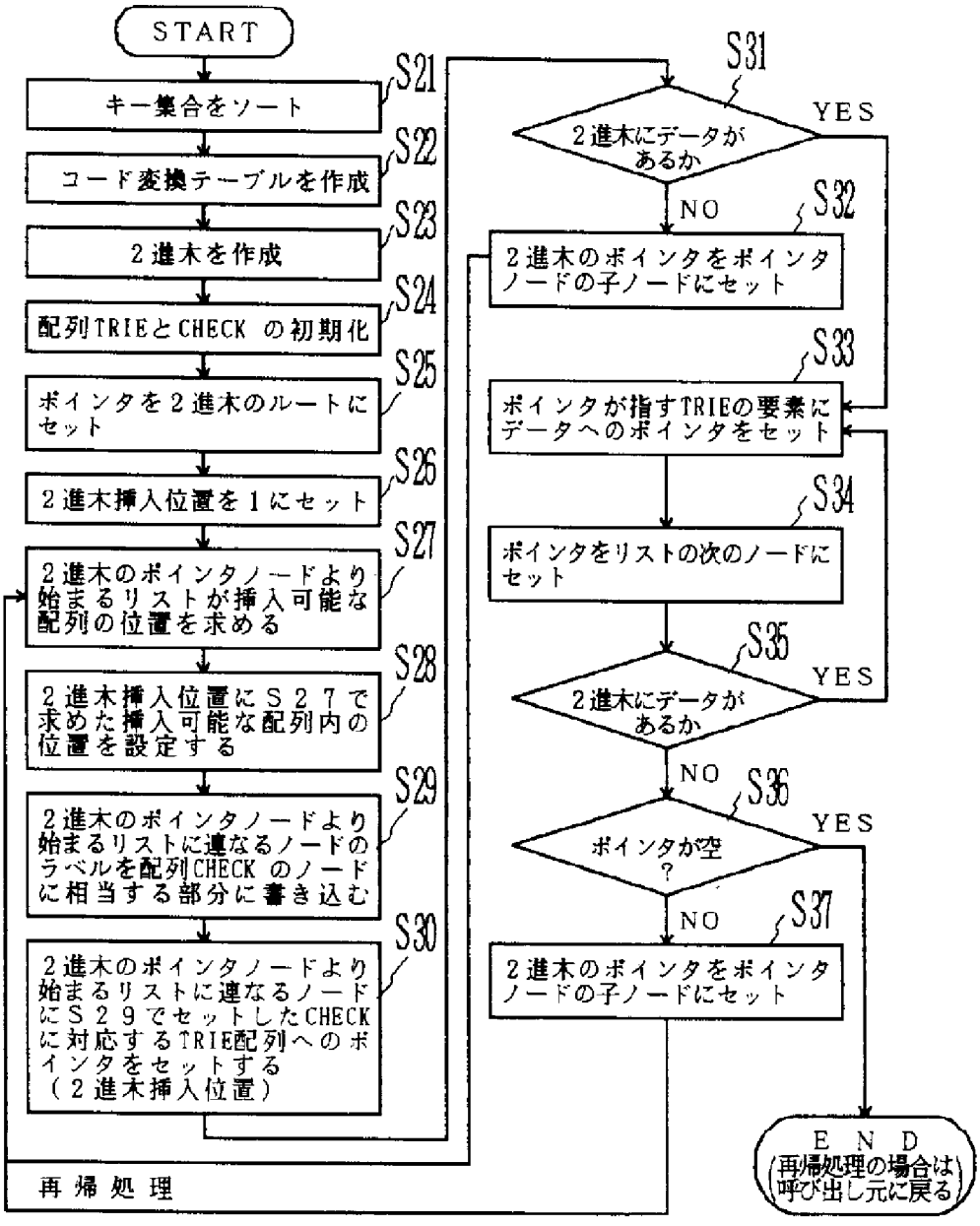
【図１３】

トライノード圧縮時の配列データの例を示す図
（その４）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	@1a	@1b	0	0	0	0	0	0	0	0
CHECK	0	0	a	b	0	0	0	0	0	0	0	0
					#	0	b	0	0	0	0	g

【図７】

トライ圧縮処理のフローチャート



【図１５】

トライノード圧縮時の配列データの例を示す図
（その６）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	@1b	0	0	0	0	0	0	0	0
CHECK	0	0	a	b	#	0	b	0	0	0	0	g

【図１６】

トライノード圧縮時の配列データの例を示す図
（その７）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	@1b	@2#	0	@2b	0	0	0	0	@2g
CHECK	0	0	a	b	#	0	b	0	0	0	0	g

【図１７】

トライノード圧縮時の配列データの例を示す図
（その８）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	@1b	K1	0	@2b	0	0	0	0	@2g
CHECK	0	0	a	b	#	0	b	0	0	0	0	g

【図１８】

トライノード圧縮時の配列データの例を示す図
（その９）

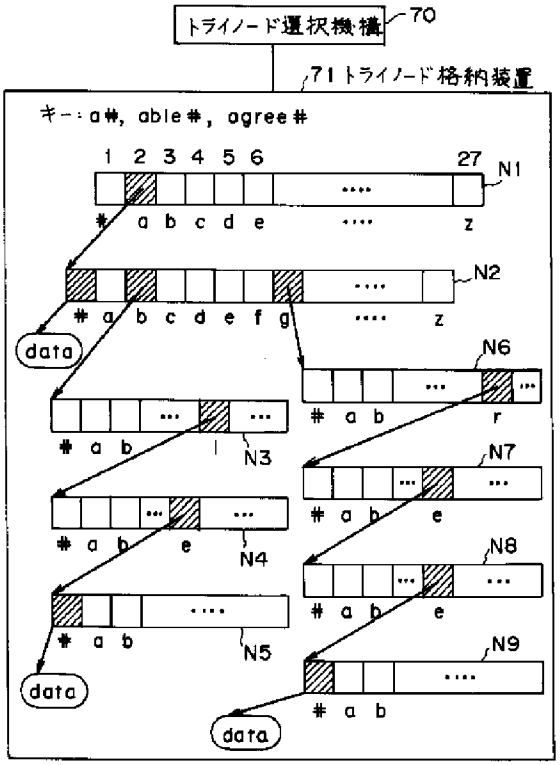
INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	@1b	K1	0	K2	0	0	0	0	K3
CHECK	0	0	a	b	#	0	b	0	0	0	0	g

【図１９】

トライノード圧縮時の配列データの例を示す図
（その10）

INDEX	1	2	3	4	5	6	7	8	9	10	11	12
TRIE	1	0	4	6	K1	0	K2	K4	0	K5	0	K3
CHECK	0	0	a	b	#	0	b	a	0	c	0	g

【図２１】



【図２０】

本発明の適用例を示す図